8p

Massachusetts Institute of Technology
Cambridge, Massachusetts
Project MAC

Artificial Intelligence Project
Memo 69

Memorandum MAC-M-159
May 29, 1964

New Language Storage Conventions

by Michael Levin

These conventions are for the implementation of the new language on a large computer on which time-sharing is the standard mode of operation. Each user is at any time assigned a certain amount of primary storage. This can be the entire memory of the machine for non time-shared operation. When this quota is filled, then it is necessary either to extend it, or to have the reclaimer routine compact the user's storage. This decision can be made at run time, and may be based on the user's storage requirements, and on the cost of primary memory at that particular instant. This may in turn depend on the degree of saturation of the system.

Primary memory is divided into public and private areas. The public area consists of system programming and library procedures. The private area allotted to any one user contains his own procedures,and data. These may be mixed in any way within his assigned storage area. There is no separation (as in LISP 1.5) into program space, list structure space, full word data space, etc.

All procedures must be coded so as not to be self-modifying. However, a procedure may have any number of arrays, or other storage allocated to it either in an own or a recursive manner. Each procedure as it is created (either from an assembly language or source language description) is coded into relocatable binary. This is always available on secondary storage with all pertinent declaration information, and with relocation information

If a user's area of primary storage is destroyed, it is possible to restore procedures merely by reading them from secondary storage, whereas if they were self-modifying, then it would be necessary to read them out as well as in. All procedures do not have to be in primary memory at execution time of a program. If a call is made to an unavailable subroutine, this user's program is hung up until the procedure can be loaded.

When a call is made to a procedure that is in the system library, it is executed from the public area of primary storage. Like all other procedures, these are not self-modifying. Storage allocation is made within the user's private area. This permits multiple use of one procedure, and even simultaneous use if the system is a multi-processor. If the protection hardware can be devised so as to permit read-only memory references to the public area of memory, then subroutine calls to the library can be made without trapping or otherwise altering the protect status.[1]

The user may specify at any time which procedures are or are not in active status. A procedure that is not in active status may not be retained in primary memory when a reclaim cycle occurs. It will be placed in active status again if it should be called from within the user's program.

---

1. An interesting way of doing this is to have three special registers that can be set only from the system monitor. One of these contains the upper bound of the public area, which starts at the bottom of the machine. Another contains the bottom of the user's area, and serves as a boundary and relocation register. The third is the upper boundary register of the user's area. The system traps any instruction that is outside the user's area and the public area, and also traps all memory modifying references within the public area. This setup permits undebugged library routines to be placed in the public area with safety, and also permits secrecy of private (but not public) storage.

The data storage conventions stated here have the following properties:

1. It is possible to obtain blocks of arbitrary length, and to abandon them when they are no longer used.

2. Lists can lead into blocks and blocks can contain lists. These two basic types of data may be mixed to any depth.

3. There is one uniform way of identifying all storage that is actually referenced by program.

4. All storage is relocatable.

5. New types of data structures may be created by supplying for each new type a code number, a marking subroutine, and a relocating subroutine. It is possible to make a data type compiler to write these subroutines, but this will not be done in the first version of the new language.

6. A restoring scheme may be used in addition to or instead of a reclaiming scheme. It is possible to have S-expressions and symmetric lists in the same storage area, and to reclaim one and restore the other.

The user's block of storage is used for his pushdown list, and as a free storage area for procedures, arrays, list structures, and any other type of data. The pushdown list starts at one end of the block, and all other data requirements are met at the other end. When the two meet, then it is necessary to have a relocating garbage collection.

Each data type has a code number associated with it. These numbers are small integers starting with 2.

S-expressions (type 2): S-expressions are referred to by pointers. If an S-expression is non-atomic, then the word that pointer refers to contains car and cdr of the S-expression in its two halves. If the S-expres-

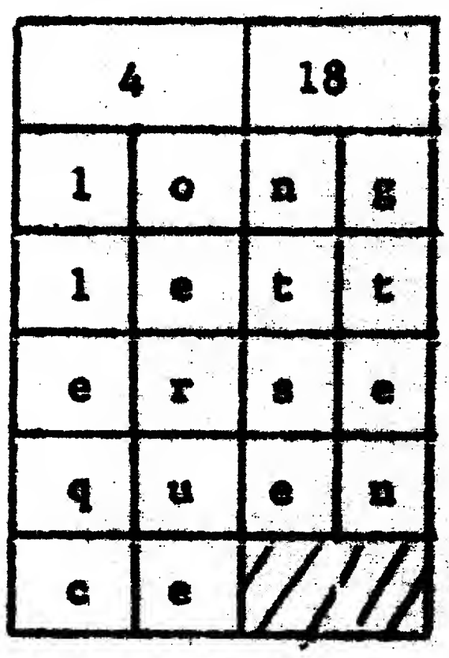
sion is atomic, then car of it will contain a small positive integer. This identifies the type of atom that it is. The number 0 is used to mark the end of a list. It behaves like the LISP 1.5 NIL, however, there is no actual atomic structure positioned at 0 in the machine.

Identifiers (type 3): The actual character representation of an identifier is not stored in the atomic structure. Instead, the atomic structure contains a key that refers to the character representation on the I/O symbol table. This structure may be placed in secondary storage when symbolic I/O is not being performed. Every atomic symbol has a property list. This can be accessed only by evaluating the primitive prop(x). The property list of x may be changed by making an assignment to prop(x).

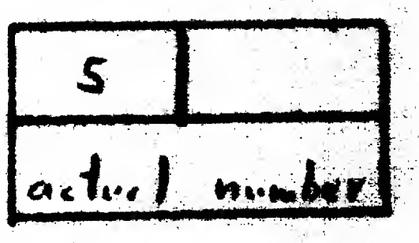
The first word of an atomic symbol contains the type (3) and the key. The next word contains the property list and a description of the remainder of the structure. If the symbol is the name of a global variable, then it is necessary to store the current values of this variable. The description would in this case contain the type of the datum to be stored there. This would inform the reclaim routine how to treat the datum, and also signal the complete size of the atomic structure. Other uses for this cell are kept open at present. If desc is 0, then the atomic structure has only two words.

3	key
desc	prop

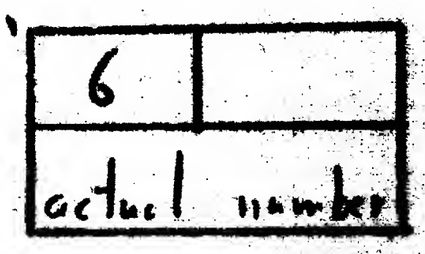
Character Strings (type 4): The datum contains the characters constituting the unquoted string and a size counter specifying the number of characters. The illustration is for the character string "longlettersequence" in a computer that stores four characters to the word.



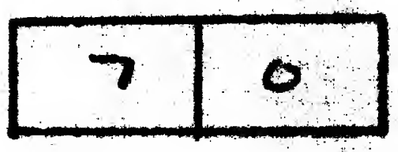
Integers (type 5):



Real Numbers (type 6):



Logical Values (type 7)

false 

true 

Arrays (type 8): Arrays make use of index tables to locate elements. The array is provided with its own subroutine for utilizing the index tables. The following example is a 4 x 5 x 6 real array as coded for a PDP-6. The effective address of an array element is obtained in accumulator 17 by loading the coordinates of the array into accumulators 1, 2, and 3, and then executing JSP 16,ARRAY+1.

```
ARRAY:  0           ,8+6B18;          (this says real array)
        MOVE 17     ,T1,1
        ADD 17      ,T2,2
        ADD 17      ,T3,3
        JUMP        ,0,16
T1:     0           ,START
        0           ,START+1
        .
        .
        .
        0           ,START+3
T2:     0           ,0
        0           ,4
        .
        .
        .
        0           ,16
T3:     0           ,0
        0           ,20
        .
        .
        .
        0           ,100              (actual data starts here)
START:
```

This convention is modified for certain types of data. For example, if double precision complex real numbers were a data type, then a cdpr array would have intervals of four on the innermost table. Logical arrays (type 8/7) are stored with the first dimension running along the word length, and occupying as many words as are necessary.

The advantages of using index tables are the speed with which random elements can be accessed, and the fact that it is not necessary to actually

move the array elements to interchange rows or columns, to transpose the array or to rotate the subscripting of a dimension. Array primitives are provided for doing all of these.

Boolean arrays (type 8/10) are stored one bit per machine word. This is the only distinction between types Boolean and Logical. The shift and rotate primitives when applied to the first dimension of a Logical array whose first dimension is exactly word length, are performed rapidly using the machine rotate or shift instructions.

Matrices (type 9): The disadvantage of the index table is that linearizing can not be used as a method of optimizing array element references. This is important for matrix multiplication, and for computing traces. Therefore, two storage conventions are offered under the names array and matrix. They appear to be the same in source language, but one or the other will be more efficient in certain circumstances.

The type matrix does not have index tables. The first word contains a 9 in the left half, and the element type in the right half. Each of the following words contains the size of a dimension, and the first one also contains the number of dimensions. The example is for a 4 x 4 x 6 symbolic matrix. The actual data consists of pointers that must be marked by the reclaim routine.

(matrix)

9	2
4	3
5	
6	

(symbolic elements)
(3 dimensions)

In the source language, array and matrix indices may occupy any integer range, e.g., -3 through +7.

The type descriptions indicated here for numerical types are for use only when the numerical type is incorporated into a symbolic expression. Individual numbers are stored directly in assigned locations on the push-down list or in global variable spaces which are similar to the LISP 1.5 special spaces.

In order to reclaim properly, it is necessary to mark all accessible data. This can be done starting from the pushdown list, and the list of all atomic symbols. When marking list structures, any occurrence of a type number indicates that the marking routine for that type should be invoked. This makes it possible for example to mix symbolic expressions and symmetric lists. As the marking proceeds, data is copied onto secondary storage, and relocation information is placed in primary storage. The compacted data is then relocated and read in. This is described in detail in A. I. Memo 58 (MAC-M-129) by Marvin Minsky.

CS-TR Scanning Project
Document Control Form

Date : 11/30/95

Report # AIM-69

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☐ Technical Report (TR) ☒ Technical Memo (TM)

☐ Other:_____

Document Information

Number of pages: 8 (12-, mACRS)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☒ Single-sided or
☐ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☒ Other: MIMEOGRAPH

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number): _____

Photographs/Tonal Material (by page number): _____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP: (1-8) UNN'ED TITLE PAGE	2-8
(9-12) SCANCONTROL, TRGT'S (3)	

Scanning Agent Signoff:
Date Received: 11/30/95 Date Scanned: 12/6/95 Date Returned: 12/7/95

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**